Available online at www.sciencedirect.com

ScienceDirect

Theoretical Computer Science 363 (2006) 234–246

Theoretical Computer Science

www.elsevier.com/locate/tcs

On size reduction techniques for multitape automata

Hellis Tamm[a, b, *, 1], Matti Nykänen[a], Esko Ukkonen[a]

[a] *Department of Computer Science, University of Helsinki, P.O. Box 68, 00014, Finland*
[b] *Institute of Cybernetics, Akadeemia tee 21, 12618 Tallinn, Estonia*

Abstract

We present a method for size reduction of two-way multitape automata. Our algorithm applies local transformations that change the order in which transitions concerning different tapes occur in the automaton graph, and merge suitable states into a single state. Our work is motivated by implementation of a language for string manipulation in database systems where string predicates are compiled into two-way multitape automata. Additionally, we present a (one-tape) NFA reduction algorithm that is based on a method proposed for DFA minimization by Kameda and Weiner, and apply this algorithm, combined with the multitape automata reduction algorithm, on our multitape automata. Empirical results on the performance of our method when applied on some multitape automata originating from string predicates are reported.

Keywords: Multitape automata; Nondeterministic finite automata; Size reduction

1. Introduction

Multitape automata, introduced by Rabin and Scott [12], are stronger but also technically subtler devices than one-tape automata. While the equivalence problem of deterministic (one-way) multitape automata is decidable [4], the same problem for nondeterministic multitape automata is not, and we are not aware of any minimization procedure for multitape automata.

In this paper, we present a method to reduce the size of two-way multitape automata. Our main motivation for this work came from the implementation of the Alignment Declaration language, a language for expressing string predicates, designed for a string handling and manipulating database system [3]. While this language provides means to declare string predicates, these declarations must be converted into an executable form to be used in database queries. As an intermediate form in this conversion, we use two-way multitape automata. To make the final executable more concise and efficient to simulate, we are interested in reducing the size of these automata. The automaton model that we consider is a modified version of the Rabin–Scott model. Our algorithm uses certain local factoring transformations that change the order in which transitions concerning different tapes occur in the automaton graph, and merge suitable states into a single state. The algorithm runs in polynomial time with respect to the number of states of the automaton.

* Corresponding author. Institute of Cybernetics, Akadeemia tee 21, 12618 Tallinn, Estonia. Tel.: +372 620 4221; fax: +372 620 4151.
E-mail addresses: hellis@cs.ioc.ee (H. Tamm), matti.nykanen@cs.helsinki.fi (M. Nykänen), esko.ukkonen@cs.helsinki.fi (E. Ukkonen).
[1] Supported by the Academy of Finland Grant 201560 and EU structural funds (RAK INNOVE project number 1.0101-0275).

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.07.027

Also, we can view these multitape automata as if they were one-tape nondeterministic automata (NFA) instead, and apply appropriate techniques to reduce their size. More specifically, we consider a method based on [11] that proposes a way to find equivalent states in an NFA and its reversal automaton, and use these equivalences to reduce the size of an NFA. The resulting reduction is at least as good as that obtained by using the right-invariant and left-invariant equivalences of [9,8]. We also apply the techniques proposed in [8] for optimally combining the right and left equivalences.

We combine this NFA reduction method along with our multitape automata size reduction algorithm into an algorithm that alternatingly applies these procedures on a given automaton until no more size reduction can be achieved. We have carried out some empirical tests in which this method was applied on several multitape automata originating from certain natural string predicates. In these tests our approach performed quite successfully.

This paper is organized as follows. Section 2 introduces the reader to the Alignment Declaration language and Section 3 describes how the string declarations are converted into multitape automata. In Section 4, we present a reduction algorithm for multitape automata, in Section 5, we consider an NFA reduction algorithm, and in Section 6, we propose a method that applies these two algorithms alternatingly on our multitape automata. Finally, in Section 7, we present experimental results.

A preliminary version of this paper appeared in [14]. Some details which are omitted in this paper can be found in [13] where we presented the multitape automata size reduction algorithm and applied it along with a minimization procedure for deterministic automata.

2. Alignment Declaration language

The motivation for the current work came from the development of the string-manipulating database system where string predicates are expressed using a specific language called Alignment Declaration language. These string expressions in the form of alignment declarations are then compiled into two-way multitape automata which are further transformed into executable programs. In the following, we give a brief introduction to the Alignment Declaration language, and in the next section we will show how the declarations are compiled into multitape automata.

The Alignment Declaration language is designed to describe string comparison and manipulation operations over several strings that are manipulated together. A basic statement of this language is an *on*-statement of the form ⟨*scan part*⟩ `on` ⟨*condition part*⟩ where both the scan and condition parts are optional. A scan part starts with a word `scan` or `rightscan` followed by a list of string variables, and its effect is to move the positions of currently considered characters of corresponding strings, respectively, to the next or the previous position. A condition part is a Boolean combination of character comparisons, such as `x='a'`, `x=y`, `x=[` or `x=]`, which evaluate true if, respectively, the current character of a string denoted by a variable `x` is `'a'`, the same as the current character of a string denoted by `y`, the left endmarker, or the right endmarker. Initially, the current character for any string is the left endmarker. An `on`-statement holds if and only if, after taking into account possible changes of current characters of the strings pointed out by the scan part, the condition part evaluates true.

An `on`-statement is an expression in the Alignment Declaration language. Other expressions are defined as follows. If Φ_1 and Φ_2 are expressions then their concatenation $\Phi_1\Phi_2$ is an expression, `repeat * times` Φ_1 `end` is an expression, and `choose` $\Phi_1|\Phi_2$ `end` is an expression. The expression $\Phi_1\Phi_2$ holds if and only if Φ_1 holds and Φ_2 holds when evaluated starting from the same currently considered character positions where the evaluation of Φ_1 ends. The expression `repeat * times` Φ_1 `end` holds if and only if a k-fold concatenation of Φ_1 with itself holds for some $k \geqslant 0$. The expression `choose` $\Phi_1|\Phi_2$ `end` holds if and only if Φ_1 holds or Φ_2 holds.

Some additional constructs are defined in the language. For example, `repeat * times scan x on x='a' end` can be written as `scan* x on x='a'`. Different string alphabets can be applied by using the `keep`-statement (see the example below). A more complete description of the language can be found in [3].

Here is an example of an alignment declaration describing a property involving two strings x and y from the alphabet $\{a, b\}$ such that y is the reversal of x:

```
reversal(x, y)
keep x in 'a', 'b'
```

```
keep y in 'a', 'b'
  scan* x on
  scan x on x=]
  repeat * times
    rightscan x on
    scan y on x=y
  end
  rightscan x on x=[
  scan y on y=]
end
end
```

3. Alignment declarations as multitape automata

In this section we discuss how the alignment declarations are translated into two-way multitape automata. First, we present the automaton model that is specially designed for our application, and then we show how the automata are obtained.

We describe the n-tape automaton model as follows. There is a window whose width is one symbol and height is n symbols, so that one symbol of each tape shows through that window at any given time. We call the showing symbol of a given tape the *current symbol* for that tape. Initially, the current symbols for all tapes are their left endmarkers. If we want to read the next symbol from a tape, we move that tape *left* with respect to the window. And if we want to read the previous symbol from a tape, we move that tape *right*. These tape movements are indicated in the automaton as transitions with the labels L_i and R_i where L and R are special symbols not belonging to the alphabet of the automaton, and i is the tape involved. For reading an input string, there are transitions with the labels like a_i where a is a symbol read from the tape $i \in \{1, \ldots, n\}$. In addition, transitions may involve special symbols `[` and `]` denoting the endmarkers, and the symbol `@` that is used to denote any string character or the right endmarker. Also, the automaton can have transitions on empty string ε.

Formally, an n-tape automaton is given by a quintuple $(Q, \Sigma, \delta, q_I, F)$ where Q is a finite set of states, Σ is the input alphabet, $\delta : Q \times (\Sigma'_{\{1,\ldots,n\}} \cup \{\varepsilon\}) \to 2^Q$ is the transition function where $\Sigma' = \Sigma \cup \{[,], @\} \cup \{L, R\}$ and $\Sigma'_{\{1,\ldots,n\}} = \{a_i \mid a \in \Sigma', i \in \{1, \ldots, n\}\}$, $q_I \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. If for some $q_1, q_2 \in Q$, $a \in \Sigma'$ and $i \in \{1, \ldots, n\}$, $q_2 \in \delta(q_1, a_i)$, then we say that there is a transition from q_1 to q_2 with label a_i, that is, with symbol a involving tape i. This transition is denoted by (q_1, a_i, q_2) or $q_1 \xrightarrow{a_i} q_2$. In case we need to use indexes to denote the symbol itself, we put the symbol in brackets like, for example, in $(a_k)_i$ where $a_k \in \Sigma'$ and $i \in \{1, \ldots, n\}$. The number of outgoing transitions of a state q is denoted by $outdegree(q)$. The number of states $|Q|$ is the *size* of the automaton.

Initially, the automaton is in the initial state. Let u be a string formed by concatenating the labels of all transitions that appear on some path in the automaton graph going from the initial state to a final state. We consider u to be an *accepting computation* if there exists an n-tuple $(w_1, \ldots, w_n)$ where $w_i \in \Sigma^*$ for $i = 1, \ldots, n$, such that if we read u from left one transition label at a time then, on seeing any c_i where $c \in \Sigma \cup \{[,]\}$ the symbol currently read from w_i is c, on seeing $@_i$ the current symbol of w_i is not the left endmarker, and on seeing L_i or R_i the current symbol of w_i is taken to be the next or the previous one, respectively. In this case, the n-tuple $(w_1, \ldots, w_n)$ is *accepted* by the automaton. The set of all n-tuples accepted by an automaton A is the *language* of A.

Now, let Φ be an alignment declaration with string variables $x_1, \ldots, x_n$. Then Φ can be translated into an n-tape automaton A as follows. First, every Boolean formula in all *on*-statements of Φ is transformed so that it consists of only `and` and `or` operations combining character comparisons in a form `x='a'`, `x=[` or `x=]`. To create A, we use a function `Compile()` described below which takes either an alignment declaration or a part of it as its first input argument and the automaton state as its second input argument, possibly creates new states and transitions into the automaton and calls itself recursively, and finally outputs an automaton state.

In the beginning, let A consist of a single final state q_F. Then, a call to the function `Compile(`Φ, q_F`)` builds up A and yields the initial state q_I of A. Let Φ_1 and Φ_2 denote either expressions in the Alignment Declaration language or parts of such Boolean formulas described above. Let q be an automaton state. Then we define the function `Compile()`

by induction over the structure of the alignment declaration as follows:

(1) `Compile(`$\Phi_1\Phi_2, q$`)` = `Compile(`Φ_1 `and` Φ_2, q`)` = `Compile(`Φ_1, `Compile(`Φ_2, q`))`;
(2) `Compile(choose` $\Phi_1|\Phi_2$ `end,` q`)` = `Compile(`Φ_1 `or` Φ_2, q`)` = q_1 where q_1 is a new state with ε-transitions to `Compile(`Φ_1, q`)` and `Compile(`Φ_2, q`)`;
(3) `Compile(repeat * times` Φ_1 `end,` q`)` = q_1 where q_1 is a new state with ε-transitions to q and `Compile(`Φ_1, q_1`)`;
(4) `Compile(on` Φ_1, q`)` = `Compile(`Φ_1, q`)`;
(5) `Compile(scan` $x_{i_1}, ..., x_{i_k}$ `on` Φ_1, q`)` = q_1 where $i_j \in \{1, ..., n\}$ and $q_1, ..., q_k$ are new states with transitions $q_j \xrightarrow{\mathtt{L}_{i_j}} q_{j+1}$ for $j = 1, ..., k$, with q_{k+1} = `Compile(`Φ_1, q`)`;
(6) `Compile(rightscan` $x_{i_1}, ..., x_{i_k}$ `on` Φ_1, q`)` = q_1 where $i_j \in \{1, ..., n\}$ and $q_1, ..., q_{2k}$ are new states with transitions $q_{2j-1} \xrightarrow{@_{i_j}} q_{2j}$, $q_{2j} \xrightarrow{\mathtt{R}_{i_j}} q_{2j+1}$, and $q_{2j-1} \xrightarrow{[_{i_j}} q_{2j+1}$ for $j = 1, ..., k$, with q_{2k+1} = `Compile(`Φ_1, q`)`;
(7) `Compile(`$x_i = \sigma, q$`)` = q_1 where $i \in \{1, ..., n\}$, $\sigma \in \Sigma \cup \{[,]\}$, and q_1 is a new state with a transition $q_1 \xrightarrow{\sigma_i} q$;
(8) `Compile(`*true*$, q$`)` = q_1 where q_1 is a new state with ε-transition to q;
(9) `Compile(`*false*$, q$`)` = q_1 where q_1 is a new state with no transitions.

Note that the compilation of an *on*-statement depends on whether its scan part starts with the word `scan` or `rightscan`. If it starts with `rightscan`, the tape movement to the right is preceded by a check whether the current character on the given tape is any string character or the right endmarker, in which case the tape move will take place; in case the current character is the left endmarker, the tape move will not occur. These checks are explicitly put into the automaton by means of corresponding transitions. If an *on*-statement starts with the word `scan`, then we could use similar reasoning and put extra transitions into the automaton. However, this is not necessary. Initially, the current character of each tape is the left endmarker in which case the tape movement to the left is possible. Those transitions in the automaton which would move the tape to the left in the situation where the current tape character is the right endmarker are replaced by ε-transitions by a later analysis of the automaton as discussed below.

The ε-transitions can be eliminated from A, using known methods from one-tape automata theory. Next, the automaton is modified to eliminate some redundant checks and tape movements from it. For this reason, the automaton is *expanded* so that it remembers in each state the last transition labels for all tapes which appeared on any path from the initial state to the given state. By using this information about labels, those transitions that can be seen as redundant are replaced with ε-transitions, and such transitions that obviously cannot be applied are eliminated from the automaton.

For example, if the last transition concerning tape i was labelled by $]_i$ and the current transition is labelled by $@_i$ then the current transition can be considered redundant and is replaced by an ε-transition. Also, a transition labelled $\mathtt{L}_i$ after a transition with the label $]_i$ is replaced by an ε-transition, and similarly for a transition labelled $\mathtt{R}_i$ after a transition with the label $[_i$. Or, if the last transition was labelled $\mathtt{L}_i$ and the current transition is labelled $[_i$ then it is obvious that the current transition is not possible to follow and it can be eliminated.

After the expansion, the ε-transitions are eliminated from the automaton. Also, the states that are not on any path from the initial state to a final state are eliminated from the automaton.

To continue with the example of Section 2, the two-tape automaton, obtained by applying the function `Compile()` on the alignment declaration `reversal(x, y)` where the ε-transitions are eliminated, is shown in Fig. 1 (left). Here, the first tape corresponds to variable x and the second one to y. The expanded automaton A_{rev} is shown in Fig. 1 (right).

4. Reduction algorithm for multitape automata

We have designed an algorithm to reduce the size of an n-tape automaton $A = (Q, \Sigma, \delta, q_1, F)$. The algorithm is based on the following four language preserving automaton transformations.

Swap Upwards: Let $q' \in Q$ be a non-initial and non-final state with $k \geqslant 1$ incoming and one outgoing transition. Let the transitions associated with q' be $q_1 \xrightarrow{(a_1)_{i_1}} q', \ldots, q_k \xrightarrow{(a_k)_{i_k}} q'$ and $q' \xrightarrow{b_j} q$, such that j refers to a tape that is different from all tapes $i_l, l \in \{1, \ldots, k\}$. Then q' and its incoming and outgoing transitions can be removed and replaced with new non-initial and non-final states $q'_1, \ldots, q'_k$ and transitions $q_1 \xrightarrow{b_j} q'_1, \ldots, q_k \xrightarrow{b_j} q'_k$ and $q'_1 \xrightarrow{(a_1)_{i_1}} q, \ldots, q'_k \xrightarrow{(a_k)_{i_k}} q$.

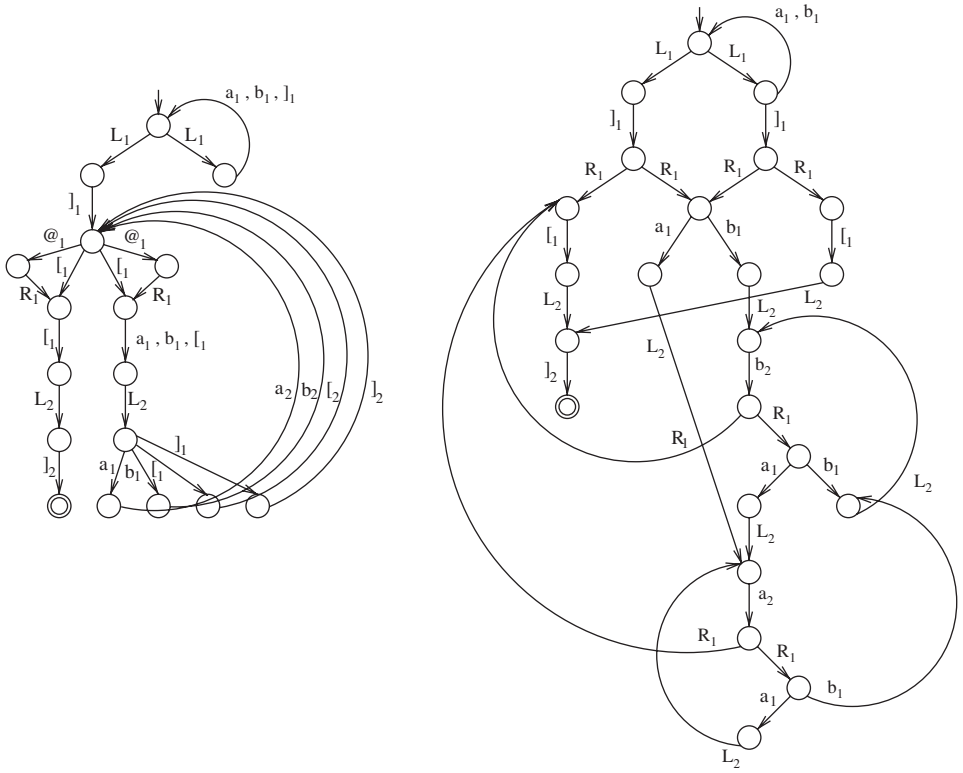

Fig. 1. The automaton corresponding to the alignment declaration `reversal(x, y)` (left) and the expanded automaton A_{rev} (right).

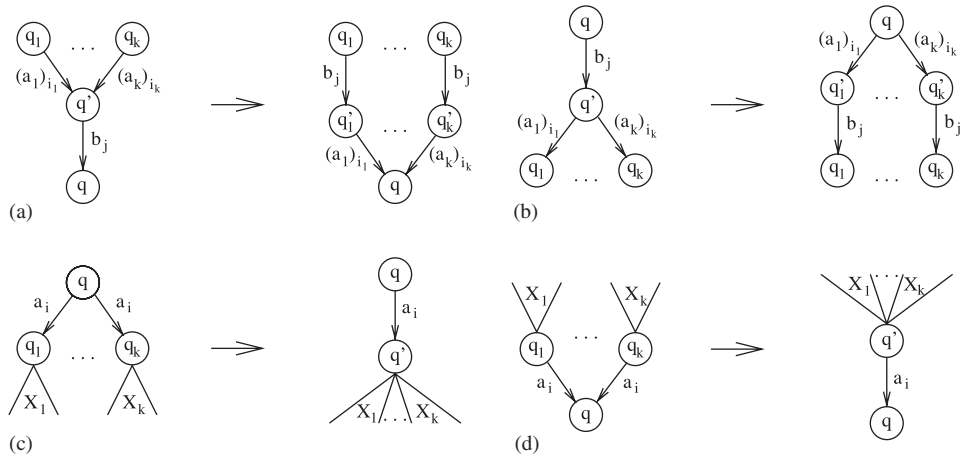

Fig. 2. Automata transformations: (a) Swap Upwards; (b) Swap Downwards; (c) Sink Combine; (d) Source Combine.

Sink Combine: Let $q_1, \ldots, q_k$ be some non-initial states of A, all having exactly one incoming transition labelled a_i from a state q of A where q is different from all q_i, $i \in \{1, \ldots, k\}$. Then $q_1, \ldots, q_k$ can be combined into one state q', meaning that $q_1, \ldots, q_k$ and their incoming and outgoing transitions are removed and replaced by a new non-initial state q' which is final if and only if any of $q_1, \ldots, q_k$ is final, with all outgoing transitions of $q_1, \ldots, q_k$ now leaving q', and the transition $q \xrightarrow{a_i} q'$.

Swap Downwards and *Source Combine* are defined symmetrically. All transformations are schematically presented in Fig. 2.

procedure MoveTransitionUp$(A, (q_1, a_i, q_2), q)$
1. **if** transition (q_1, a_i, q_2) exists in A **then**
2. use *Sink Combine* transformation to merge all such states that are reachable from q_1 by a transition labelled by a_i and suitable for this transformation;
3. **if** $q \neq q_1$ and $outdegree(q_1) = 1$ **then**
4. use *Swap Upwards* transformation on the out-transition of q_1 and let T be a set of transitions labelled by a_i created by this transformation;
5. **for all** $(q_1', a_i, q_2') \in T$ where $q_1', q_2' \in Q$ **do**
6. `MoveTransitionUp`$((q_1', a_i, q_2'), q)$;

Fig. 3. Procedure `MoveTransitionUp()`.

Definition 1. Let $q, q', q'' \in Q, a \in \Sigma'$, and $i \in \{1, \ldots, n\}$. A transition $q' \xrightarrow{a_i} q''$ of A is called a *future transition* for the state q and tape i if and only if there exists a path $q \xrightarrow{(b_1)_{j_1}} q_1 \xrightarrow{(b_2)_{j_2}} q_2 \ \ldots \ q_{k-1} \xrightarrow{(b_k)_{j_k}} q_k$ in A such that $q' = q_{k-1}$, $q'' = q_k$, $a = b_k$, $i = j_k$ and for $l = 1, \ldots, k-1$, $j_l \neq i$.

A central part of the reduction algorithm is the procedure `MoveTransitionUp()` presented in Fig. 3. Let us fix some $q \in Q, a \in \Sigma'$ and $i \in \{1, \ldots, n\}$. We want to find a set of future transitions for q and i, with the label a_i, such that by calling `MoveTransitionUp()` for each of these transitions and the state q, we can reduce the number of states of A by a certain amount. When this procedure is invoked with a transition (q_1, a_i, q_2) and the state q, its goal is to decrease the number of states and transitions of A by "moving" the transition (q_1, a_i, q_2) in the automaton graph "up", applying *Swap Upwards* and *Sink Combine* transformations on the way, until this transition, along with one or more other transitions which have the same label, will be replaced by a transition out of q (instead of q_1).

Now, we will specify a set of conditions which guarantee that applying the procedure `MoveTransitionUp()` is possible and leads to the reduction of the number of states and transitions of the automaton.

Let us denote a set of future transitions for q and i bearing the label a_i, by $ft_{q,i,a}$, and let us denote the set of all paths in A, which start from q and end by any transition in $ft_{q,i,a}$, by $P_{ft_{q,i,a}}$. Consider the following conditions imposed on the path set $P_{ft_{q,i,a}}$. Let p be a path in the set $P_{ft_{q,i,a}}$ and let the two last states on p be q' and q''. Then the conditions are as follows:

(i) there are no loops in p, except that q'' may be equal to q;
(ii) every state on p that appears after q and before q'' is non-initial and non-final, all of its incoming and outgoing transitions are traversed by some path in $P_{ft_{q,i,a}}$, and all of its incoming transitions involve a tape that is different from i;
(iii) if q' has more than one outgoing transition then q'' is non-initial and has only one incoming transition.

Now, the following propositions hold:

Proposition 2. *There is a unique maximal set $FT_{q,i,a}$ of future transitions for q and i, with the label a_i, such that the conditions* (i)–(iii) *hold for the set $P_{FT_{q,i,a}}$.*

Proof. Consider the set $ft^{\text{all}}_{q,i,a}$ of all future transitions for q and i, with the label a_i. If $ft^{\text{all}}_{q,i,a}$ is an empty set then the proposition is trivially true, with $FT_{q,i,a}$ being empty as well. Now, let us assume that the set $ft^{\text{all}}_{q,i,a}$ is not empty. Then we partition $ft^{\text{all}}_{q,i,a}$ into non-intersecting non-empty subsets $ft_1, \ldots, ft_k$ in the following way. Consider any two transitions t and u in $ft^{\text{all}}_{q,i,a}$ with the corresponding path sets $P_{\{t\}}$ and $P_{\{u\}}$ consisting of paths starting from q and ending by t and u, respectively. Let t and u belong to the same subset ft_j, $j \in \{1, \ldots, k\}$, if and only if there exists a pair of paths $p_t \in P_{\{t\}}$ and $p_u \in P_{\{u\}}$ such that p_t and p_u have a common state that is different from the starting and the ending states of both p_t and p_u.

For every $j \in \{1, \ldots, k\}$, let us consider the set of paths P_{ft_j} that start from q and end by any transition belonging to ft_j. It is easy to see by the definition of the sets ft_j that if the conditions (i)–(iii) hold for some path sets

$P_{ft_{j_1}}, \ldots, P_{ft_{j_l}}$ where $j_m \in \{1, \ldots, k\}, m \in \{1, \ldots, l\}$, then these conditions are also true for the union of these path sets.

But if a set of future transitions contains as a subset a non-empty proper subset ft'_j of ft_j but not the whole ft_j, then the corresponding set of paths, beginning from q and ending by such a transition, does not satisfy the condition (ii). This is because then there must be a state r on a path starting from q and ending by some transition in ft'_j such that some out-transition of r does not belong to this path whereas it belongs to some path from q ending by some transition in ft_j.

To summarize, a maximal subset of $ft^{\text{all}}_{q,i,a}$ such that the conditions (i)–(iii) hold for all paths starting from q and ending by any transition in this subset is a union of all ft_j such that the conditions (i)–(iii) hold for P_{ft_j}. This set is uniquely defined. □

Proposition 3. *The series of calls to the procedure* `MoveTransitionUp()` *where it is invoked with every transition in* $FT_{q,i,a}$ *and* q*, results in size reduction of* A *by* $|FT_{q,i,a}| - 1$ *states. Also, at least the same number of transitions are eliminated from* A *by this process.*

Proof. Consider the series of calls to the procedure `MoveTransitionUp()` as specified in the proposition. Let us denote by $FT'_{q,i,a}$ the set of transitions, which initially consists of all transitions in $FT_{q,i,a}$, and which is modified according to the changes that the above-mentioned calls to the procedure `MoveTransitionUp()` produce in the automaton. That is, those transitions in $FT'_{q,i,a}$ that are removed from the automaton by Sink Combine and Swap Upwards transformations are also removed from $FT'_{q,i,a}$, and all new transitions with the label a_i that are created by the same transformations to replace the removed transitions are added to the set $FT'_{q,i,a}$. Using the same notation as above, let $P_{FT'_{q,i,a}}$ be the set of all paths in A which start from q and end by any transition belonging to $FT'_{q,i,a}$. Based on changes that the Sink Combine and Swap Upwards transformations can make in the automaton, it is not difficult to see that the conditions (i)–(iii) hold for all paths in $P_{FT'_{q,i,a}}$ after any number of Sink Combine and/or Swap Upwards transformations have been performed during the `MoveTransitionUp()` calls under consideration.

When the procedure `MoveTransitionUp()` is called for a given transition $(q_1, a_i, q_2) \in FT'_{q,i,a}$ and the state q, it first checks if the transition still exists, and if yes, then it uses the Sink Combine transformation to merge such states reachable from q_1 by a transition labelled by a_i that satisfy the conditions for this operation. If there are at least two such states to merge then, by (iii), this merging concerns every state that is reachable from q_1 by any transition in $FT'_{q,i,a}$.

If q_1 is different from q and if there is only one transition leaving q_1 then by (ii), the Swap Upwards transformation can be applied to this transition, followed by a set of recursive calls to `MoveTransitionUp()` for transitions labelled by a_i that were created by the Swap Upwards transformation. In case $q_1 \neq q$ and q_1 has more than one outgoing transition, all of these transitions belong to (one or more) paths in $P_{FT'_{q,i,a}}$ each of which end by a transition belonging to $FT'_{q,i,a}$. When considering these transitions forming a subset of $FT'_{q,i,a}$, we claim that when `MoveTransitionUp()` is called for each of the transitions in this subset, then these calls to `MoveTransitionUp()` finally eliminate all outgoing transitions of q_1 and replace them by a single transition for which the Swap Upwards transformation can be applied. This is because, during this process, there is always some transition in the above-mentioned subset of $FT'_{q,i,a}$, for which either Sink Combine or Swap Upwards transformation can be applied, or otherwise some of the conditions (i)–(iii) cannot hold.

The conditions (i)–(iii) guarantee that the process involving the series of calls to `MoveTransitionUp()` as stated in the proposition concerns only the states that are on some path of $P_{FT'_{q,i,a}}$, terminates, and replaces the transitions in $FT_{q,i,a}$ by a single transition originating from q with the label a_i (although q can still have other outgoing transitions with this label that do not belong to $FT_{q,i,a}$).

During this process, $|FT_{q,i,a}| - 1$ states and at least as many transitions are eliminated, as shown next. When the Sink Combine transformation is applied to merge some states reachable from q_1 by a transition labelled by a_i then along with the mergeable states it eliminates the same number of transitions originating from q_1. Also, as the merged state (as well as any other state) may have at most one transition with any label to any state, those outgoing transitions of the merged state that would otherwise become duplicates are eliminated as well. The Swap Upwards transformation creates the equal number of new states and transitions with the label a_i going to these states, thus increasing the number of

states and transitions by the same amount. Therefore, as totally $|FT_{q,i,a}|$ transitions are replaced by a single transition, $|FT_{q,i,a}| - 1$ states and at least as many transitions are eliminated by the process. □

Let us suppose that we find the maximal sets of future transitions for a state q and tape i, for all possible labels, such that the corresponding path set of each of these transition sets satisfies the conditions (i)–(iii). The next proposition ensures that the application of the series of `MoveTransitionUp()` calls as in Proposition 3, for each of these sets, is independent of the order in which the sets are handled.

Proposition 4. *Let $q \in Q$, $a, b \in \Sigma'$, and $i \in \{1, \ldots, n\}$. Let $FT_{q,i,a}$ and $FT_{q,i,b}$ be the maximal sets of future transitions for q and i, labelled a_i and b_i, respectively, with their corresponding path sets $P_{FT_{q,i,a}}$ and $P_{FT_{q,i,b}}$ which satisfy the conditions* (i)–(iii) *as in Proposition* 2. *Let us first apply the transformations described in Proposition* 3 *for the set $FT_{q,i,a}$. After that, Proposition* 3 *still holds for the set $FT_{q,i,b}$.*

Proof. First, we claim that for any path pair $p_t \in P_{FT_{q,i,a}}$ and $p_u \in P_{FT_{q,i,b}}$, if p_t and p_u have a common state other than q then it is the ending state of both p_t and p_u. Indeed, if we suppose that there is a common state r on paths p_t and p_u such that $r \neq q$ and r is not the ending state of p_t or p_u, then the condition (ii) must be violated. Therefore, such state cannot exist.

Based on this observation, the transformations performed for the set $FT_{q,i,a}$ as described in Proposition 3 do not interfere with the transformations for the set $FT_{q,i,b}$. Therefore, the proposition holds. □

Similarly to the conditions (i)–(iii), symmetric conditions can be specified that allow to eliminate automaton states by applying a procedure that uses the *Source Combine* and *Swap Downwards* transformations.

The reduction algorithm (presented in Fig. 4) uses a variable m to indicate the number of states eliminated from A. The idea of the algorithm is that for each tape of A, as many states as possible are eliminated from A using the procedure `Upwards()`, and from its copy A_1 using a symmetric procedure `Downwards()` (not shown). Given the automaton tape *tape*, `Upwards()` finds for each state q a set $FT_{q,\text{tape}}$ that is the union of all maximal sets $FT_{q,\text{tape},a}$ of future transitions for q and *tape*, with some symbol a, such that the conditions (i)–(iii) are satisfied for the path set $P_{FT_{q,\text{tape},a}}$ (as in Proposition 2). For all $FT_{q,\text{tape},a}$ that consist of at least two transitions, a state q' is found which has the same set of future transitions $FT_{q',\text{tape},a}$ for this tape and symbol that satisfies Proposition 2, and which is as close to the transitions in $FT_{q,\text{tape},a}$ as possible. Then the procedure `MoveTransitionUp()` is called for all of the transitions in $FT_{q',\text{tape},a}$ and q', and by Proposition 3, the value of m is increased by $|FT_{q',\text{tape},a}| - 1$. After considering every such set $FT_{q,\text{tape},a}$, the loop over all states is started again. This process continues until no further reductions of A can be achieved using this approach for any state of A. The return value of `Upwards()` indicates the number of states eliminated by it. The procedure `Downwards()` acts similarlyin a symmetric fashion.

In case any states were eliminated from either A or A_1, a smaller one of these automata is retained and the next round with a next tape is performed using two copies of that automaton. Also, the value of m is updated accordingly. This process is continued until no more states are eliminated for any tape.

To complete this section, we show that the reduction algorithm runs in polynomial time with respect to the number of states of the automaton. Let us denote $N = |Q|$ and $S = |\Sigma'|$.

Proposition 5. *The reduction algorithm runs in* $O(n^3 S^3 N^4)$ *time.*

Proof. Copying A into another automaton can be done in $O(nSN^2)$ time as there are at most N states and nSN^2 transitions in A.

In the procedure `Upwards()`, finding a set $FT_{q,\text{tape}}$ (line 5) is done as follows. A depth-first search in the automaton graph starting from the state q by following its out-transitions, until reaching every future transition for q and *tape*, is performed. To sort out the individual sets $FT_{q,\text{tape},a}$ of future transitions for each involved label a which satisfy the conditions (i)–(iii) as in Proposition 2, a special marking of automaton states is used. The marking that indicates whether the paths from q to the future transitions of q and *tape* satisfy the conditions (i)–(iii) is performed by the above-mentioned depth-first search and by extra searches to propagate the marking appropriately via incoming and outgoing transitions in situations where (i)–(iii) do not hold. As these searches involve traversing each transition of A at most a constant number

procedure Upwards$(A, tape)$
1. $m := 0$; $reduced := true$;
2. **while** $reduced = true$ **do**
3. $\quad reduced := false$;
4. $\quad$ **for all** $q \in Q$ as long as $reduced = false$ **do**
5. $\quad\quad$ find a set $FT_{q,tape} = \bigcup_{a \in \sigma' \subseteq \Sigma'} FT_{q,tape,a}$ such that for each $a \in \sigma'$, $FT_{q,tape,a}$ is as in Proposition 2;
6. $\quad\quad$ **for all** $a \in \sigma'$ where $|FT_{q,tape,a}| > 1$ **do**
7. $\quad\quad\quad$ find a state q' such that $FT_{q',tape,a} = FT_{q,tape,a}$, $FT_{q',tape,a}$ is as in Proposition 2, and the longest path from q' to the originating state of any transition in $FT_{q,tape,a}$ is of minimal length;
8. $\quad\quad$ **for all** $t \in FT_{q',tape,a}$ **do**
9. $\quad\quad\quad$ `MoveTransitionUp`(A, t, q');
10. $\quad\quad m := m + |FT_{q',tape,a}| - 1$;
11. $\quad\quad reduced := true$;
12. return m;

Reduce A
1. $m := 0$; $A_1 :=$ `CopyOf`(A); $reduced := true$;
2. **while** $reduced = true$ **do**
3. $\quad reduced := false$;
4. $\quad$ **for** $tape := 1$ **to** n **do**
5. $\quad\quad m_{up} :=$ `Upwards`$(A, tape)$;
6. $\quad\quad m_{down} :=$ `Downwards`$(A_1, tape)$;
7. $\quad\quad$ **if** $m_{up} > 0$ **or** $m_{down} > 0$ **then**
8. $\quad\quad\quad$ **if** $m_{up} \geq m_{down}$ **then**
9. $\quad\quad\quad\quad A_1 :=$ `CopyOf`(A);
10. $\quad\quad\quad\quad m := m + m_{up}$;
11. $\quad\quad\quad$ **else**
12. $\quad\quad\quad\quad A :=$ `CopyOf`(A_1);
13. $\quad\quad\quad\quad m := m + m_{down}$;
14. $\quad\quad\quad reduced := true$;
15. return A, m;

Fig. 4. Procedure `Upwards()` and the reduction algorithm.

of times, they take totally $O(nSN^2)$ time. Finding a state q' (line 7) can be achieved in $O(N)O(nSN^2) = O(nSN^3)$ time. A series of `MoveTransitionUp()` calls with all transitions in $FT_{q',\text{tape},a}$ takes $O(n^2S^2N^3)$ time, as at most $O(nSN)$ work is done with each of the $O(nSN^2)$ transitions of A by the Sink Combine and Swap Upwards transformations. Thus, the `for` loop of `Upwards()` of lines 4–11 takes time $O(N)O(nSN^2) + O(S)O(n^2S^2N^3) = O(n^2S^3N^3)$. Totally, this loop is run for $O(nN)$ times because there can be $O(N)$ such loop runs that reduce the size of the automaton and for each such run there can be $O(n)$ runs that do not result in reduction.

As a similar reasoning can be applied to the procedure `Downwards()`, and as the total running time of the `if` command in lines 7–14 of the main algorithm is $O(N)O(n)O(nSN^2) = O(n^2SN^3)$, the total time taken by the reduction algorithm is $O(n^3S^3N^4)$. □

Corollary 6. *For a fixed number of tapes and fixed alphabet, the time complexity of the reduction algorithm is* $O(N^4)$.

5. Reducing the size of an NFA

Our multitape automata can also be viewed as (one-tape) NFAs over the alphabet $\Sigma'_{\{1,\ldots,n\}}$. Therefore, we can consider applying NFA size reduction methods on our multitape automata as well.

Let $A = (Q, \Sigma, \delta, I, F)$ be an NFA with the state set Q, the alphabet Σ, the transition function $\delta : Q \times \Sigma \to 2^Q$, the set of initial states $I \subseteq Q$, and the set of final states $F \subseteq Q$. The transition function δ is extended to $\delta : 2^Q \times \Sigma^* \to 2^Q$ so that $\delta(P, a) = \bigcup_{q \in P} \delta(q, a)$, $\delta(P, \varepsilon) = P$, and $\delta(P, ax) = \delta(\delta(P, a), x)$ for all $P \subseteq Q$, $x \in \Sigma^*$, and $a \in \Sigma$, with ε being the empty string. The set $L(A) = \{x \in \Sigma^* \mid \delta(I, x) \cap F \neq \emptyset\}$ is the *language* accepted by A. Let $p, q \in Q$. Then the set $L_{\mathrm{L}}(A, q) = \{x \in \Sigma^* \mid q \in \delta(I, x)\}$ is the *left language* of q and the set $L_{\mathrm{R}}(A, q) = \{x \in \Sigma^* \mid \delta(q, x) \cap F \neq \emptyset\}$ is the *right language* of q. The states p and q are called *equivalent* if and only if $L_{\mathrm{R}}(A, p) = L_{\mathrm{R}}(A, q)$. The *reversal* of A is the automaton $A^{\mathrm{r}} = (Q, \Sigma, \delta^{\mathrm{r}}, F, I)$ where $\delta^{\mathrm{r}}(p, a) = \{q \mid p \in \delta(q, a)\}$ for all $p \in Q$ and $a \in \Sigma$. A special case of an NFA is a *deterministic* automaton (DFA) which has a unique initial state and for which $|\delta(q, a)| \leqslant 1$ for every $q \in Q$ and $a \in \Sigma$.

Since the NFA minimization problem is PSPACE-complete [10], algorithms that reduce the size of NFAs but which do not necessarily produce minimal results, are of interest. Although the DFA minimization can be done efficiently, the conversion of an NFA into DFA by the subset construction [6] can possibly result in an exponentially larger automaton.

Ilie and Yu [9] considered the right-invariant and left-invariant equivalence relations to reduce the size of NFAs. They defined $\equiv_{\mathrm{R}}$ as the coarsest equivalence relation over Q satisfying the following two conditions:

(1) $\equiv_{\mathrm{R}} \cap (F \times (Q - F)) = \emptyset$,
(2) $\forall p, q \in Q, \forall a \in \Sigma, (p \equiv_{\mathrm{R}} q \Rightarrow \forall q' \in \delta(q, a), \exists p' \in \delta(p, a), q' \equiv_{\mathrm{R}} p')$.

The equivalence $\equiv_{\mathrm{R}}$ is the largest equivalence over Q which is right-invariant with respect to A [9]. Note that $\equiv_{\mathrm{R}}$ is a refinement of the commonly known equivalence relation between automaton states which is based on the right languages of states as defined earlier. If $\equiv_{\mathrm{R}}$ is given, we could reduce the automaton by merging all states in each equivalence class and modifying the transitions correspondingly. Symmetrically, the left-invariant equivalence relation $\equiv_{\mathrm{L}}$ is defined using the reversal automaton A^{r}. The automaton A can be reduced using either $\equiv_{\mathrm{R}}$ or $\equiv_{\mathrm{L}}$ or some combination of both equivalences. The result of applying these equivalences depends on the order in which the equivalences are used. Recently, in [8] an efficient algorithm was given to optimally use the combination of the right and left equivalences for NFA reduction.

Champarnaud and Coulon [1,2] used preorders to reduce the size of NFAs. In [7], efficient algorithms were presented for computing equivalences and preorders.

Here we consider a method for NFA reduction based on the theory by Kameda and Weiner [11]. This method specifies a way to find equivalent states in the automaton A and its reversal A^{r}. This method is exponential as it makes use of the subset construction but we think it is of interest because of its simplicity.

Let A be an NFA and let C be an automaton obtained from A^{r} by the subset construction. That is, any state of C is a subset of the state set of A. By Kameda and Weiner [11], two states of A are equivalent if and only if they appear exactly in the same states of C. They mention that this is useful for DFA minimization. Namely, if A is a DFA then by merging the equivalent states one can find a minimal DFA. In the case of A being an NFA, this method can be used for the size reduction of the automaton although the result is not necessarily a minimal NFA. Similarly, we can find the equivalent states of the reversal automaton. Let B be an automaton obtained by applying the subset construction on A. Then, two states of A^{r} are equivalent if and only if they appear exactly in the same states of B. By the appropriate merging of the equivalent states of A^{r}, we can reduce the size of A^{r} (and use this to reduce A).

Polynomially computable right-invariant and left-invariant equivalences $\equiv_{\mathrm{R}}$ and $\equiv_{\mathrm{L}}$ of [9,8] are refinements of the above state equivalence relations for A and A^{r}. Thus, the reductions according to the above equivalences result in automaton size reduction by at least the same amount (or more) as obtained by [9,8].

Similarly to [8], we can possibly get a smaller NFA by combining the reductions corresponding to the two equivalences above. We briefly describe the approach taken in [8] in the following. Consider an NFA with the state set Q. Both the right and left equivalences of the NFA define a partition of Q into equivalence classes. Let these partitions be $\Pi_{\mathrm{R}} = \{X_1, \ldots, X_r\}$ and $\Pi_{\mathrm{L}} = \{X_{r+1}, \ldots, X_{r+s}\}$, respectively. In the reduction of the NFA, the states in the same equivalence class are merged into a single state. Let the reduction be $X^* = \{X_1^*, \ldots, X_l^*\}$, where each X_i^* is a set of equivalent states that are merged into a single state. That is, the reduced automaton has l states. For each $i \in \{1, \ldots, l\}$ there exists some $\pi(i) \in \{1, \ldots, r + s\}$, so that $X_i^* \subseteq X_{\pi(i)}$. As $\bigcup_{i=1}^{l} X_i^* = Q$, also $\bigcup_{i=1}^{l} X_{\pi(i)} = Q$, and $\{X_{\pi(1)}, \ldots, X_{\pi(l)}\}$ is a *set cover* for Q. An optimal solution for the instance $\langle Q, X = \Pi_{\mathrm{R}} \cup \Pi_{\mathrm{L}} \rangle$ of the set covering problem where the goal is to find the smallest subset of X that includes all elements of Q would specify an optimal use of equivalences. Let S^* be such a smallest set cover for $\langle Q, X \rangle$. To obtain the reduced NFA using the equivalences optimally, first, duplicated occurrences of the same state are removed from S^*, and then all the states in the same subset $S_i \in S^*$ are merged into

a single state. This set covering problem can be modelled as a bipartite graph $G_B = (L \cup R, E)$. The vertexes of this graph are all sets X_i, $i = 1, \ldots, r + s$, and edges correspond to the states of the NFA, such that each edge connects the two sets that contain the state. A minimum vertex cover for G_B corresponds to an optimal set cover for $\langle Q, X \rangle$. A minimum vertex cover can be easily derived (as shown in [8]) from a maximum matching which can be computed using the algorithm of Hopcroft and Karp [5].

To find an optimal use of the state equivalences computed for A and A^r by the method based on [11] in order to reduce the automaton A we take mainly the same approach as in [8]. We just replace the right-invariant and left-invariant equivalences by the "normal" state equivalences for A and A^r. That is, now the partition $\Pi_R = \{X_1, \ldots, X_r\}$ consists of the equivalent state sets of A and $\Pi_L = \{X_{r+1}, \ldots, X_{r+s}\}$ consists of the equivalent state sets of A^r.

We also add an extra optimization step that is based on the following two propositions.

Proposition 7. *Let $X' = \{X_{i_1}, \ldots, X_{i_u}\} \subseteq \Pi_R$ and $X'' = \{X_{r+j_1}, \ldots, X_{r+j_v}\} \subseteq \Pi_L$ be such that for all $i_k \in \{i_1, \ldots, i_u\}$, $X_{i_k} \cap X_{r+j_m} \neq \emptyset$ for all $j_m \in \{j_1, \ldots, j_v\}$ and $X_{i_k} \cap X_{r+j_n} = \emptyset$ for all $j_n \in \{1, \ldots, s\} \setminus \{j_1, \ldots, j_v\}$. Then all states that belong to the sets X_{i_k} of the collection X' can be merged into one single state without changing the language of the NFA.*

Proof. We could merge each set X_{i_k} of states into a single state p_{i_k}, $i_k \in \{i_1, \ldots, i_u\}$. Then we notice that $L_L(A, p_{i_k}) = \bigcup_{j_m \in \{j_1, \ldots, j_v\}} \{L_L(A, q) \mid q \in X_{r+j_m}\}$. That is, all states $p_{i_1}, \ldots, p_{i_u}$ have the same left language and thus can be merged into a single state. □

Proposition 8. *Let $X' = \{X_{i_1}, \ldots, X_{i_u}\} \subseteq \Pi_R$ and $X'' = \{X_{r+j_1}, \ldots, X_{r+j_v}\} \subseteq \Pi_L$ be such that for all $j_m \in \{j_1, \ldots, j_v\}$, $X_{i_k} \cap X_{r+j_m} \neq \emptyset$ for all $i_k \in \{i_1, \ldots, i_u\}$ and $X_{i_t} \cap X_{r+j_m} = \emptyset$ for all $i_t \in \{1, \ldots, r\} \setminus \{i_1, \ldots, i_u\}$. Then all states that belong to the sets X_{r+j_m} of the collection X'' can be merged into one single state without changing the language of the NFA.*

Proof. Similar to the proof of Proposition 7. □

Based on Propositions 7 and 8, our optimization step can be described as follows. We form a matrix with r rows and s columns so that it contains a row for each set X_i, $i = 1, \ldots, r$, and a column for each set X_{r+j}, $j = 1, \ldots, s$. The (i, j) entry is 1 if $X_i \cap X_{r+j} \neq \emptyset$, otherwise the entry is 0. Then we replace all sets X_i that have an identical pattern of 1's and 0's in the corresponding rows, by the union of these sets in Π_R. Similarly, we replace all sets X_{r+j} that have an identical pattern of 1's and 0's in the corresponding columns, by the union of these sets in Π_L. Since the number of sets in the collections Π_R and Π_L decreases by these replacements, a minimum vertex cover for the bipartite graph formed from the modified Π_R and Π_L can be smaller than a minimum cover for the original graph. Therefore, we can possibly (although not necessarily) get a better reduction for an NFA applying this optimization.

Next, we proceed with forming a bipartite graph as described above, find a maximum matching and a minimum vertex cover for the graph, remove duplicate occurrences of the states from the sets that form the vertex cover, and finally merge the states in the sets of the cover to obtain a reduced NFA.

It is possible that after we have reduced the size of an NFA this way, applying the same method on the reduced NFA results in a further size reduction. Therefore, we repeatedly apply this method on the reduced NFA, until no more reduction is achieved.

6. Reducing the size of a multitape automaton

To continue with the example of Sections 2 and 3, if we apply the NFA reduction algorithm presented in Section 5 to the automaton A_{rev}, then its size is reduced from 23 states to 11 states. The resulting automaton denoted by $Red_{NFA}(A_{rev})$ is shown in Fig. 5 (left).

Now, applying the multitape automata reduction algorithm of Section 4 after the NFA reduction can lead to a further size reduction of the automaton. If we apply this algorithm to $Red_{NFA}(A_{rev})$ then the result is the automaton $Red_{multi}(Red_{NFA}(A_{rev}))$ having nine states as shown in Fig. 5 (right). Further application of the NFA reduction algorithm on this automaton does not make it any smaller.

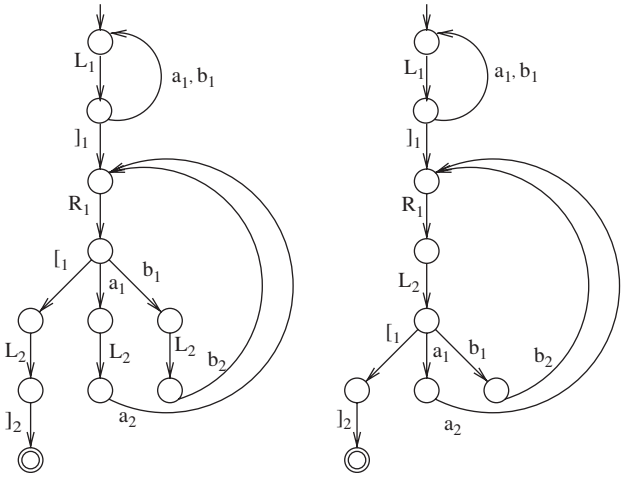

Fig. 5. The automata $Red_{\mathrm{NFA}}(A_{\mathrm{rev}})$ (left) and $Red_{\mathrm{multi}}(Red_{\mathrm{NFA}}(A_{\mathrm{rev}}))$ (right).

String predicate	n	$\lvert\Sigma\rvert$	$\lvert A_{orig}\rvert$	$\lvert A_{exp}\rvert$	Automaton size during the reduction process			
reversal	2	2	17	23	Red_{NFA} 11	Red_{Multi} 9	Red_{NFA} 9	
					Red_{multi} 16	Red_{NFA} 11	Red_{multi} 9	Red_{NFA} 9
substring	2	2	11	18	9	9		
					17	9	9	
subsequence	2	2	11	17	7	7		
					16	7	7	
prefix	2	2	9	16	7	7		
					15	7	7	
suffix	2	2	18	25	11	11		
					22	11	11	
concatenation	3	2	21	20	13	12	12	
					19	13	12	12
shuffle	3	2	21	51	12	10	10	
					45	12	10	10
overlap	3	2	15	48	21	20	20	
					44	20	19	19
edit distance	3	4	24	168	28	27	27	
					143	28	27	27

Fig. 6. Automata sizes before and during the reduction process.

On the other hand, we can also reduce A_{rev} by applying the multitape automata reduction algorithm first. The resulting automaton (not shown) has 16 states. Now, if we apply the NFA reduction procedure on this automaton, the result is the same as the automaton $Red_{\mathrm{NFA}}(A_{\mathrm{rev}})$. Further reduction of this automaton is as above.

In this example, the end result of applying these two algorithms one after another does not depend on which of them was applied first. However, generally, this is not the case.

Finally, we propose the following algorithm to reduce the size of a multitape automaton A that alternatingly applies two size-reducing algorithms. Apply two sequences of algorithms consisting of the NFA reduction procedure of Section 5 and the multitape automata reduction algorithm of Section 4 by turn on A, at one time starting with the NFA reduction algorithm and the other time starting with the multitape automata reduction algorithm, and stopping when no more size reduction occurs to A. Output the smaller of the resulting two automata.

7. Experimental results

To test the algorithm presented at the end of Section 6, we have considered a set of alignment declarations expressing different string properties, and made experiments with the corresponding automata. The results of the experiments are presented in the table in Fig. 6. For each string predicate, the table shows the number of tapes n and the alphabet size $|\Sigma|$, the size of the original automaton $|A_{\mathrm{orig}}|$ (the result of applying the function `Compile()` on the corresponding alignment declaration) after eliminating ε-transitions from it, and the size of the expanded automaton $|A_{\mathrm{exp}}|$ after ε-transition elimination. The reduction algorithm is applied on the ε-transition-free expanded automaton A_{exp} of each string predicate. The table shows the size of the automaton during the reduction process, given in two rows: the upper row shows the automaton size in the sequence where the NFA reduction algorithm is applied first, and the lower row shows the automaton size in the sequence where the multitape automata reduction algorithm is applied first. The numbers in the columns with Red_{NFA} and Red_{multi} indicate the size of the automaton in the reduction process, after applying the NFA reduction or the multitape automata reduction algorithm, respectively. The $Red_{\mathrm{NFA}}/Red_{\mathrm{multi}}$ pattern is explicitly shown only for the first string predicate, for other predicates the pattern is similar.

Even if both of these reduction sequences end up with the automata of the same size, the automata may be different. For most cases in our experiments, the reduced automaton is smaller than the original one, although this is not always so, as for automata of *overlap* and *edit distance* predicates. However, if one has in mind the efficiency of simulating the computations of automata, then avoiding redundant checks of tape symbols and those paths that are not possible to follow seem to be important. Fortunately, most of the size growth in the expanded automata seems to disappear as the result of the reduction process.

References

[1] J.-M. Champarnaud, F. Coulon, NFA reduction algorithms by means of regular inequalities, Theoret. Comput. Sci. 327 (2004) 241–253.
[2] J.-M. Champarnaud, F. Coulon, Erratum to "NFA reduction algorithms by means of regular inequalities", Theoret. Comput. Sci. 347 (2005) 437–440.
[3] G. Grahne, R. Hakli, M. Nykänen, H. Tamm, E. Ukkonen, Design and implementation of a string database query language, Inform. Systems 28 (2003) 311–337.
[4] T. Harju, J. Karhumäki, The equivalence problem of multitape finite automata, Theoret. Comput. Sci. 78 (1991) 347–355.
[5] J.E. Hopcroft, R. Karp, An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs, SIAM J. Comput. 2 (4) (1973) 225–231.
[6] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979.
[7] L. Ilie, G. Navarro, S. Yu, On NFA reductions, in: Theory is Forever, Lecture Notes in Computer Science, Vol. 3113, Springer, Berlin, 2004, pp. 112–124.
[8] L. Ilie, R. Solis-Oba, S. Yu, Reducing the size of NFAs by using equivalences and preorders, in: Proc. of the CPM 2005, Lecture Notes in Computer Science, Vol. 3537, Springer, Berlin, 2005, pp. 310–321.
[9] L. Ilie, S. Yu, Reducing NFAs by invariant equivalences, Theoret. Comput. Sci. 306 (2003) 373–390.
[10] T. Jiang, B. Ravikumar, Minimal NFA problems are hard, SIAM J. Comput. 22 (6) (1993) 1117–1141.
[11] T. Kameda, P. Weiner, On the state minimization of nondeterministic automata, IEEE Trans. Comput. C-19 (7) (1970) 617–627.
[12] M.O. Rabin, D. Scott, Finite automata and their decision problems, IBM J. Res. Develop. 3 (1959) 114–125.
[13] H. Tamm, On minimality and size reduction of one-tape and multitape finite automata, Ph.D. Thesis, Department of Computer Science, University of Helsinki, Finland, 2004.
[14] H. Tamm, M. Nykänen, E. Ukkonen, Size reduction of multitape automata, in: Proc. of the CIAA 2005, Lecture Notes in Computer Science, Vol. 3845, Springer, Berlin, 2006, pp. 307–318.